# Laboratory work nr. 2 part 2

Scober, Mihai, IS-231M

In this part of the laboratory I need to

- a) repeat it first steps from part 1 for new csv file;
- b) repeat that for many other regression models;
- c) find the best one, tune, try to explain;

</ul>

I will start with straight copy pasting 8 steps for new dataset

In [28]:
```
# This is second part, I need to repeat 8 first steps with new dataset
# imports
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
from pipelinehelper import PipelineHelper
```

In [29]:
```
# get dataset
dataset = pd.read_csv('Data1.csv')
print(dataset.head(10))
```

```
   Sample code number  Clump Thickness  Uniformity of Cell Size  \
0             1000025                5                        1
1             1002945                5                        4
2             1015425                3                        1
3             1016277                6                        8
4             1017023                4                        1
5             1017122                8                       10
6             1018099                1                        1
7             1018561                2                        1
8             1033078                2                        1
9             1033078                4                        2

   Uniformity of Cell Shape  Marginal Adhesion  Single Epithelial Cell Size  \
0                         1                  1                            2
1                         4                  5                            7
2                         1                  1                            2
3                         8                  3                            3
4                         1                  3                            2
5                        10                  8                            7
6                         1                  1                            2
7                         2                  1                            2
8                         1                  1                            2
9                         1                  1                            2

   Bare Nuclei  Bland Chromatin  Normal Nucleoli  Mitoses  Class
0            1                3                1        1      2
1           10                3                2        1      2
2            2                3                1        1      2
3            4                3                7        1      2
4            1                3                1        1      2
5           10                9                7        1      4
6           10                3                1        1      2
7            1                1                1        5      2
8            1                2                1        1      2
9            1                2                1        1      2
```

In [30]:
```
# got a wise advice to actually START using np array instead of dataframes.
X = np.array(dataset.drop(['Class', axis= 1))
y = np.array(dataset['Class'])
```

In [31]:
```
# another advice I got is to train and fit the model before splitting it into train and test variables
standard_scaler = StandardScaler()
scaled_matrix = standard_scaler.fit_transform(X)
```

In [32]:
```
# split into train and test data
X_train, X_test, y_train, y_test = train_test_split(scaled_matrix, y, test_size=0.2)
```

In [33]:
```
# create SVC, make a prediction
svc_classifier = SVC(kernel='linear', random_state = 0)
svc_classifier.fit(X_train, y_train)
svc_prediction = svc_classifier.predict(X_test)
svc_prediction
# when I was exactly the way to see the results
```

Out[33]:
```
array([2, 4, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 4, 4, 2, 2, 2, 2,
       2, 2, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 4, 2,
       2, 2, 4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 4, 2, 2, 2,
       2, 2, 4, 4, 2, 4, 4, 2, 2, 4, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2,
       2, 2, 4, 4, 4, 2, 4, 4, 4, 4, 2, 2, 4, 2, 2, 2, 2, 4, 4, 2,
       4, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 4, 2, 4, 2, 4, 4,
       2, 2, 2, 2, 2], dtype=int64)
```

In [34]:
```
# compare prediction with actual results
print(confusion_matrix(y_test,svc_prediction))
print(classification_report(y_test,svc_prediction))
# results differ from one predict to another, but precision is bigger than recall most of the times
# and in general looks good enough
#
# Also this is worth mentioning that without 'fit and transform' function before splitting data into
# train and test I was getting worse results than here
print(accuracy_score(y_test, svc_prediction))
```

```
[[92  0]
 [ 0 45]]
              precision    recall  f1-score   support

           2       1.00      1.00      1.00        92
           4       1.00      1.00      1.00        45

    accuracy                           1.00       137
   macro avg       1.00      1.00      1.00       137
weighted avg       1.00      1.00      1.00       137

1.0
```

In [35]:
```
# now with cross validate the result to see if we can reach more accuracy
scores = cross_val_score(svc_classifier, scaled_matrix, y, cv=10)
print(scores)
print(scores.mean())
# Yph, it seems we were just very lucky
```

```
[0.92753623 0.97101449 0.95652174 0.94117647 0.98529412 0.97058824
 0.95588235 1.         0.97058824 0.98529412]
0.9646859903381642
```

It already looks that this branch is much more fitting for model training and making predictions, but let's try other models too

In [36]:
```
# next step is grid search
# set C parameters and kernel
param_grid = [
    {'C': [0.25, 0.5, 0.75, 1], 'kernel': ['linear','rbf']}
]

grid_search_result = GridSearchCV(svc_classifier, param_grid, scoring="accuracy", n_jobs= -2)
grid_search_result.fit(X_train, y_train)

print("Best parameters set found on development set: "+str(grid_search_result.best_params_))
print()
print("Grid scores on development set:")
print()
means = grid_search_result.cv_results_['mean_test_score']
stds = grid_search_result.cv_results_['std_test_score']
best_result = ()
mean_zero = 0
# zip to loop through two numerical lists
for mean, std, params in zip(means, stds, grid_search_result.cv_results_['params']):
    if(mean_zero < mean):
        mean_zero = mean
        best_result['mean'] = mean
        best_result['params'] = params
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
print("Best results found with %0.3f mean and parameters %r" % (best_result['mean'], best_result['params']))
# it is fun, all 3 rbf kernels showed the same result
print("Classification report:")
print()
y_true, y_pred = y_test, grid_search_result.predict(X_test)
print(classification_report(y_true, y_pred))
```

```
Best parameters set found on development set: {'C': 0.75, 'kernel': 'rbf'}

Grid scores on development set:

0.956 (+/-0.056) for {'C': 0.25, 'kernel': 'linear'}
0.956 (+/-0.033) for {'C': 0.25, 'kernel': 'rbf'}
0.956 (+/-0.052) for {'C': 0.5, 'kernel': 'linear'}
0.956 (+/-0.042) for {'C': 0.5, 'kernel': 'rbf'}
0.956 (+/-0.052) for {'C': 0.75, 'kernel': 'linear'}
0.956 (+/-0.042) for {'C': 0.75, 'kernel': 'rbf'}
0.956 (+/-0.052) for {'C': 1, 'kernel': 'linear'}
0.958 (+/-0.042) for {'C': 1, 'kernel': 'rbf'}

Best results found with 0.960 mean and parameters {'C': 0.75, 'kernel': 'rbf'}
Classification report:

              precision    recall  f1-score   support

           2       1.00      1.00      1.00        92
           4       1.00      1.00      1.00        45

    accuracy                           1.00       137
   macro avg       1.00      1.00      1.00       137
weighted avg       1.00      1.00      1.00       137
```

This accuracy looks almost the same as what we attempted before, so there is nothing to be surprised about, I guess. It is in general outstanding, no matter what parameters we use.

Now I need to implement a lot of different models

1. Logistic Regression
2. Decision Tree Classifier
3. Random Forest Classifier (with nb_trees = 10)
4. K- Nearest Neighbors (K-NN)
5. Naive Bayes
6. Support Vector Machine (SVM)

In [37]:
```
# 1. logistic regression
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)
prediction = logistic_regression.predict(X_test)
scores = cross_val_score(logistic_regression, scaled_matrix, y, cv=10)
print("%0.2f accuracy with a std of %0.2f" % (scores.mean(), scores.std()))
```

```
0.97 accuracy with a std of 0.02
```

Making regression models one by one like what you see above is cute, but I want to make the code optimized, so here is my generic approach:

In [38]:
```
best_result = {}
mean_zero = 0
def generic_regression_model_execution(regression_model, regression_model_name):
    global mean_zero
    global best_result
    regression_model.fit(X_train, y_train)
    prediction = regression_model.predict(X_test)
    print(regression_model_name," : confusion matrix ")
    print(confusion_matrix(y_test, prediction))
    print(regression_model_name," : classification report ")
    print(classification_report(y_test, prediction))
    scores = cross_val_score(regression_model, scaled_matrix, y, cv=10)
    mean = scores.mean()
    std = scores.std()
    if(mean_zero < mean):
        mean_zero = mean
        best_result['regression_model_name'] = regression_model_name
        best_result['accuracy'] = mean
        best_result['std'] = std
    print(regression_model_name," : %0.3f accuracy with std of %0.3f" % (mean, std))
    print()
    print("**************************************************")
    print()

list_of_regression_models = []
list_of_regression_models['LogisticRegression'] =LogisticRegression()
list_of_regression_models['DecisionTreeClassifier'] =DecisionTreeClassifier()
list_of_regression_models['RandomForestClassifier'] =RandomForestClassifier(n_estimators=10)
list_of_regression_models['K-NearestNeighbors'] =KNeighborsClassifier()
list_of_regression_models['NaiveBayes'] =GaussianNB()
list_of_regression_models['SupportVectorMachine'] =SVC()
for key, value in list_of_regression_models.items():
    generic_regression_model_execution(value, key)
print("Best regression model is %s with accuracy %0.3f and std %0.3f" % (best_result['regression_model_name'],
                                                                         best_result['accuracy'],
                                                                         best_result['std']))
```

```
LogisticRegression  : confusion matrix
[[92  0]
 [ 1 44]]
LogisticRegression  : classification report
              precision    recall  f1-score   support

           2       0.99      1.00      0.99        92
           4       1.00      0.98      0.99        45

    accuracy                           0.99       137
   macro avg       0.99      0.99      0.99       137
weighted avg       0.99      0.99      0.99       137

LogisticRegression  : 0.966 accuracy with std of 0.023

==========================================================

DecisionTreeClassifier  : confusion matrix
[[91  1]
 [ 4 41]]
DecisionTreeClassifier  : classification report
              precision    recall  f1-score   support

           2       0.94      0.99      0.96        92
           4       0.98      0.87      0.92        45

    accuracy                           0.95       137
   macro avg       0.96      0.93      0.94       137
weighted avg       0.95      0.95      0.95       137

DecisionTreeClassifier  : 0.947 accuracy with std of 0.026

==========================================================

RandomForestClassifier  : confusion matrix
[[92  0]
 [ 3 42]]
RandomForestClassifier  : classification report
              precision    recall  f1-score   support

           2       0.97      1.00      0.98        92
           4       1.00      0.93      0.97        45

    accuracy                           0.98       137
   macro avg       0.98      0.97      0.97       137
weighted avg       0.98      0.98      0.98       137

RandomForestClassifier  : 0.961 accuracy with std of 0.023

==========================================================

K-NearestNeighbors  : confusion matrix
[[92  0]
 [ 1 44]]
K-NearestNeighbors  : classification report
              precision    recall  f1-score   support

           2       0.99      1.00      0.99        92
           4       1.00      0.98      0.99        45

    accuracy                           0.99       137
   macro avg       0.99      0.99      0.99       137
weighted avg       0.99      0.99      0.99       137

K-NearestNeighbors  : 0.966 accuracy with std of 0.024

==========================================================

NaiveBayes  : confusion matrix
[[91  1]
 [ 1 44]]
NaiveBayes  : classification report
              precision    recall  f1-score   support

           2       0.99      0.99      0.99        92
           4       0.98      0.98      0.98        45

    accuracy                           0.99       137
   macro avg       0.98      0.98      0.98       137
weighted avg       0.99      0.99      0.99       137

NaiveBayes  : 0.962 accuracy with std of 0.020

==========================================================

SupportVectorMachine  : confusion matrix
[[92  0]
 [ 0 45]]
SupportVectorMachine  : classification report
              precision    recall  f1-score   support

           2       1.00      1.00      1.00        92
           4       1.00      1.00      1.00        45

    accuracy                           1.00       137
   macro avg       1.00      1.00      1.00       137
weighted avg       1.00      1.00      1.00       137

SupportVectorMachine  : 0.966 accuracy with std of 0.020

==========================================================

Best regression model is LogisticRegression with accuracy 0.966 and std 0.023
```

Here I would say way of choosing the best model is not fully correct. I made my choice based on accuracy and std. But logistic model had at least one wrong result, where the support vector machine had 0 and similar accuracy. With 0.3f it is rounded, but in general logistic model somehow has more accuracy than SVM. In case that I made the decision based on missed predictions, I would definitely pick the SVM one. 0 misses, amazing.

And once again, this dataset works for all models very nice.

In [54]:
```
# My best model is logistic one, so now is the time to tune it's parameters
param_grid = [
    {'classifier' : [LogisticRegression()],
     'classifier__penalty' : ['l1', 'l2'],
     'classifier__C' : np.logspace(-4, 4, 50),
     'classifier__solver' : ['liblinear']}
]

# Create grid search object

clf = GridSearchCV(pipe, param_grid = param_grid, cv = 10, verbose=True, n_jobs=-2)

# Fit on data

best_clf = clf.fit(X_train, y_train)
best_prediction = best_clf.predict(X_test)
best_scores = cross_val_score(best_clf.best_estimator_, scaled_matrix, y, cv=10)
print("%0.3f accuracy with std %0.3f" % (best_scores.mean(), best_scores.std()))
print("tuned hyperparameters :(best parameters) ",best_clf.best_params_)
print("accuracy :",best_clf.best_score_)
```

```
Fitting 10 folds for each of 50 candidates, totalling 500 fits
0.972 accuracy with std 0.018
tuned hyperparameters :(best parameters)  {'classifier': LogisticRegression(C=0.012742749857031334, solver='liblinear'), 'classifier__C': 0.012742749857031334, 'classifier__penalty': 'l2', 'classifier__solver': 'liblinear'}
accuracy : 0.9671043771043771
```

Since I trust the internet, I took my parameters I could find there and pasted them here. With some tweaking grid search decided that logistic model is always around 0.96 anyway, so I will not pick any parameters here as important ones. Most of results I saw were almost the same.

Moreover, as a conclusion I am surprised that logistic model 'won', I personally thought that SVM will fit the best. To be fair, all the models showed great results, and probably if missed shots mattered more than accuracy I would still pick SVM model. I am not that experienced with machine learning to play with parameters and to make some strong assumptions, so I'll just say that this time the dataset was the most important factor why the results are so good, not the models or my coding skills.